Modeling Turnpike: a Model-Driven Framework for Domain-Specific Software Development[*]

Hiroshi Wada
Advisor: Junichi Suzuki

Department of Computer Science
University of Massachusetts, Boston
hiroshi_wada@otij.org and jxs@cs.umb.edu

Abstract. This paper overviews the Modeling Turnpike (mTurnpike) project, which investigates a generic model-driven development framework that supports various domain-specific solutions (i.e. modeling, programming and development process to directly deal with domain concepts). This paper identifies a series of research issues to create such a framework, and describes how the mTurnpike project addresses those issues. The proposed framework allows developers to model and program domain concepts (as UML models and attribute-oriented programs) and to transform them to compilable code. This paper also describes future directions of the mTurnpike project.

1. Introduction

Modeling technologies have matured to the point where they can offer significant leverage in all aspects of software development. Given modern modeling technologies, the focus of software development has been shifting from implementation technology domains toward the concepts and semantics in problem domains. The more directly application models can represent domain concepts, the easier it becomes to specify applications. A key goal of modeling technologies is to map modeling concepts directly to domain concepts [1].

Domain Specific Language (DSL) is a promising solution to directly represent and implement domain concepts [2]. DSLs are visual or textual languages targeted to particular problem domains, rather than general-purpose languages that aim at any software problems. Various DSLs have been proposed and used in academic, industrial and government communities. Although several experience reports have demonstrated DSLs can improve software development productivity (e.g. [3]), existing DSLs are supported only by specific tools and frameworks; there are few generic frameworks supporting arbitrary DSLs.

This Ph.D. research investigates a generic model-driven development (MDD) framework that supports various domain-specific solutions (i.e. modeling, programming and development processes to directly deal with domain concepts), and empirically evaluates a series of techniques to develop such a framework. Steps towards creating the proposed framework include investigating a generic foundation to handle arbitrary DSLs; strategies, principles and tradeoffs in different DSL designs (e.g. DSL syntax and semantics); building blocks for modeling and programming domain concepts; transformation strategies from domain concepts to the final (compilable) source code; development processes to

[*] This research is supported in part by OGIS International, Inc. and Electric Power Development Co., Ltd.

leverage the proposed framework; model-driven approaches for maintenance and tests; and performance implications of major functional components in the framework.

This project addresses these research issues through implementing the proposed framework, and empirically evaluates the impact of framework designs on various evaluation criteria such as versatility, flexibility, productivity, ease of use, and maintainability.

2. Modeling Turnpike: The Proposed Framework

This project proposes and investigates a new MDD framework, called Modeling Turnpike (or mTurnpike). mTurnpike allows developers to model and program domain concepts in arbitrary DSLs and to transform them to the final (compilable) source code in a seamless manner [4, 5]. Leveraging the notions of UML metamodeling and attribute-oriented programming, mTurnpike provides an abstraction to represent domain concepts at the modeling and programming layers simultaneously.

At the modeling layer, domain concepts are represented as a *Domain Specific Model (DSM)*, which is a set of UML 2.0 diagrams (Fig. 1). At the programming layer, domain concepts are represented as a *Domain Specific Code (DSC)*, which consists of attribute-oriented programs (Fig. 1). Attribute-oriented programming is a program marking technique [6]. Programmers can *mark* program elements (e.g. classes and methods) to indicate that they maintain domain-specific semantics. By hiding the implementation details of those semantics from program code, attributes increase the level of programming abstraction and reduce programming complexity, resulting in simpler and more readable programs. The program elements associated with attributes are transformed to more detailed (compilable) programs by a supporting tool (e.g. pre-processor). mTurnpike uses the annotations in Java 2 Standard Edition 5.0 as the syntax of attributes.

mTurnpike consists of the frontend and backend systems (Fig. 1). The frontend system is implemented as DSC Generator. It transforms domain concepts from the modeling layer to programming layer, and vise versa, by providing a seamless mapping between DSMs and DSCs. Each DSL is specified as a UML profile, which extends the UML standard metamodel to define stereotypes and tagged-values, in order to express domain concepts. Given a DSL, a DSM is represented as a set of UML 2.0 class and composite struc-

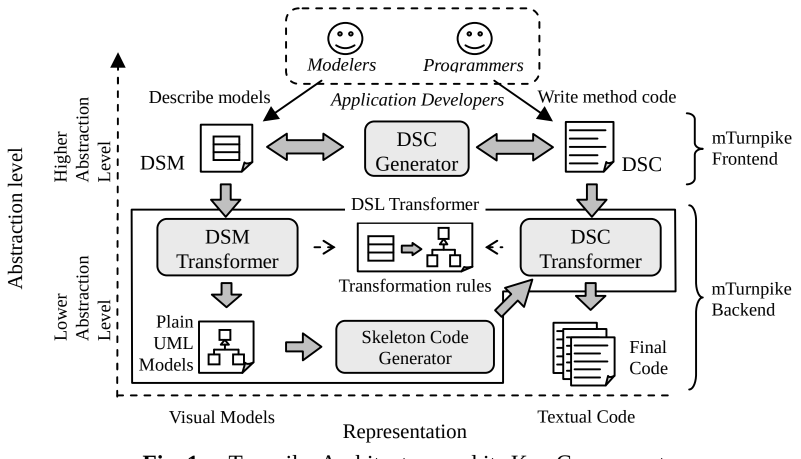

Fig. 1. mTurnpike Architecture and its Key Components.

ture diagrams[1]. Each DSC consists of Java interfaces and classes decorated with annotations[2]. After DSC Generator generates a DSC (i.e. annotated code), programmers write method code in the generated DSC in order to implement dynamic behaviors for domain concepts[3]. DSC Generator maps the stereotypes and tagged-values in a DSM to the attributes (annotations) in a DSC, and vise versa.

The backend system of mTurnpike transforms a DSM and DSC into a more detailed model and program by applying a given transformation rule. mTurnpike allows developers to define arbitrary transformation rules, each of which specifies how to specialize a DSM and DSC to particular implementation and deployment technologies. For example, a transformation rule may specialize them to a database system, while another rule may specialize them to a remoting system. The backend system consists of DSM Transformer, Skeleton Code Generator and DSC Transformer (Fig. 1).

DSM Transformer accepts a DSM, and specializes it to a particular implementation and deployment technologies. Given a transformation rule, it transforms (or unfolds) the DSM model elements associated with stereotypes and tagged-values into the plain UML model elements that do not have them (Fig. 1). Skeleton Code Generator takes a plain UML model created by DSM Transformer, and generates skeleton Java code that corresponds to the input plain UML model (Fig. 1). DSC Transformer accepts a DSC generated by DSC Generator, method code written on the generated DSC by programmers, and skeleton code generated by Skeleton Code Generator. Then, DSC Transformer combines them to generate the final compilable code (in Java). DSC Transformer extracts method code embedded in an input DSC, and copies the method code to an input skeleton code. DSC Transformer analyses a given transformation rule (Fig. 1) in order to determine where each method code is copied in an input skeleton code.

Currently, mTurnpike is implemented in Java. The implementation of its frontend system is completed, and its backend system is in an early stage of implementation.

3. Research Issues and Contributions

This section summarizes how this project addresses the research issues identified in Section 1.

- ***A generic foundation to handle arbitrary DSLs.*** As described in Section 1, mTurnpike is expected to be generic to handle arbitrary DSLs at both the modeling and programming layers. For this purpose, mTurnpike adopts a *language-in-language* design strategy in which different specialized languages are defined on top of a generic and customizable language. In mTurnpike, each DSL is defined as a UML profile. This means that DSMs are derivative models defined on top of UML. As descried in Section2, mTurnpike maps the stereotpes and tagged-values in a DSM and the annotations in a DSC. DSCs (i.e. annotated code) are derivative programs defined on top of Java. mTurnpike allows developers to define arbitrary DSLs and use arbitrary DSMs and DSCs. This design strategy contributes to improve the versatility of mTurnpike. In

[1] This work is one of the first attempts to exploit UML 2.0 to define and use DSLs.

[2] This work is the first attempt to bridge the gap between UML modeling and attribute-oriented programming.

[3] The methods in generated DSCs are empty because DSMs and DSCs specify static structure of domain concepts.

order to demonstrate how to exploit mTurnpike in application development, it has been used to develop distributed systems with a DSL for Service-Oriented Architecture (SOA) [4, 5]. The SOA DSL abstracts distributed systems using two major domain-specific concepts, *service interface* and *connections between services*, and hides the details of implementation and deployment technologies (e.g. programming languages and remoting systems). The SOA DSL allows users to specify, as DSMs, connection semantics (e.g. queuing and secure connections), message invocation semantics (e.g. synchronous and asynchronous invocations) and message filtering semantics (e.g. message conversion and aggregation).

- ***Building blocks for modeling and programming domain concepts.*** It is an important issue in MDD tools how to describe domain concepts at the modeling and programming layers. Traditional MDD tools accept domain-specific models in UML or other notations, and generate skeleton source code in general-purpose languages (e.g. Java, C++ and C#) [7, 8, 9]. In this scheme, programmers and modelers work at different abstraction levels. Although modelers work on modeling at a higher abstraction level, programmers have to work on generated code (e.g. writing method code) at a lower abstraction level. The generated source code is often hard to read and understand. It is also complicated, time consuming and error-prone to modify and extend the generated code. mTurnpike allows both modelers and programmers to work at a higher abstraction level (Fig. 1). Programmers write method code in DSCs (i.e. annotated code) before generating skeleton source code. This means that programmers can focus on coding application's business logic without handling the details in implementation and deployment technologies. Also, DSCs (i.e. annotated code) are much more readable and easier to understand than the skeleton source code (in general-purpose languages) generated by traditional MDD tools.
- ***Transformation strategies from domain-specific models to the final (compilable) source code.*** It is another important issue in MDD tools how to transform models (or domain-specific models) toward the final compilable source code [10]. Using a set of transformation rules, mTurnpike transforms a DSM to a DSC and a plain UML model (i.e. a model that has stereotypes and tagged-values), and combines the DSC and plain UML model to generate the compilable code. mTurnpike allows developers to define transformation rules in a declarative manner. Declarative transformation rules are more readable and easier to write and maintain than procedural ones. This framework design contributes to improve ease of use and maintainability in mTurnpike.
- ***Development processes to leverage the proposed framework.*** mTurnpike maps domain concepts between the modeling and programming layers in a seamless and bi-directional manner. This mapping allows modelers and programmers to deal with the same set of domain concepts in different representations (i.e. UML models and annotated code), yet at the same level of abstraction. Thus, modelers do not have to involve programming details, and programmers do not have to possess detailed domain knowledge and UML modeling expertise. This separation of concerns can reduce the complexity in application development, and increase the productivity for developers to model and program domain concepts.
- ***Performance Implications of major functional components.*** Since mTurnpike employs a new development style to implement domain concepts, it is not clear how

much overhead and memory space is necessary to execute major functional components in mTurnpike. Preliminary measurement results show that the frontend system of mTurnpike (i.e. DSC Generator) has fairly small footprint (up to 5MB) and works efficiently (up to 5 seconds) to process a mid-size application that contains 100 classes each of which has 10 data fields, two stereotypes and 10 tagged-values [5].

4. Future Directions

This section summarizes the future directions of this project.

- ***A generic foundation to handle arbitrary DSLs.*** mTurnpike currently supports only one DSL for each transformation from a DSM to compilable code. However, an application may require two or more DSLs, e.g. DSLs for a vertical domain and a horizontal domain. A vertical domain means a business specific domain, such as banking and manufacturing. A horizontal domain means a technology specific domain, such as a remoting system. The SOA DSL described in Section 3 is an example of horizontal DSL. A future work will enhance mTurnpike to support at least three hosizontal/vertical DSLs.
- ***Strategies, principles and tradeoffs in different DSL designs.*** mTurnpike currently uses UML and UML profile to define DSLs. Although UML is a standard and powerful modeling language, it requires developers to understand its complex metamodel to define DSLs (UML profiles). Also, UML is designed along with the object-oriented paradigm. To define a non-OO model, developers need to create a new metamodel using MOF (i.e. meta-metamodel) [11]. Several MDD initiatives provide DSLs based on non-UML modeling languages targeted to particular domains (e.g. data modeling and user interface). These languages are usually easier to customize than UML, but they are proprietary and may raise a learning curve. Tradeoffs and selection criteria between them are not well explored yet. A future work will address this research issue through extending mTurnpike to support MOF and support non-UML modeling notations such as BPMN [12].
- ***Transformation strategies from domain-specific models to the final (compilable) source code.*** mTurnpike currently supports only static structure of a domain-specific models. Dynamic behavior is defined as method code in DSCs. mTrunpike copies method code from a DSC to the final code, and it does not transform the code. A future work will address a code transformation mechanism between a DSC and the final code. Also, to make the code transformation mechanism generic (i.e. supporting arbitrary programming languages), a future work will investigate a generic meta-model for programming languages (e.g. code transformation mechanism based on EBNF [13]).
- ***Model-driven approaches for maintenance and tests.*** mTurnpike currently does not provide complete level of traceability between a DSM and the final code, and a model-driven mechanism for debugging and testing. Currently, developers write method code in DSCs, but it is not compilable. If errors occur in the final code, developers have to revise method code in a DSC and transform it again to the final code. This round trip process may impose a burden on developers. A future work will ad-

dress a mechanism to directly debug DSCs in a model-driven manner and maintain traceability between DSMs and the final code.

- ***Performance Implications of major functional components.*** A set of preliminary performance measurements reveal mTurnpike's forntend has several bottlenecks. A future work will address performance improvements on them, and reveal performance implications of major functional components in mTrunpike's backend.

5. Conclusion

This paper overviews the current status and future directions of the mTurnpike project. The author of the paper started this project as his preliminary Ph.D. research in May 2004, and enrolled in the Ph.D. program of University of Massachusetts, Boston in September 2005. His graduation is expected in 2010.

References

1.. G. Booch, A Brown, S Iyengar, J. Rumbaugh and B. Selic, "An MDA Manifesto," In *D. Frankel and J. Parodi (eds.), The MDA Journal: Model Driven Architecture Straight from the Masters,* Chap. 11, Meghan-Kiffer, Dec. 2004.
2. S. Cook, "Domain-Specific Modeling and Model-driven Architecture," In *D. Frankel and J. Parodi (eds.), The MDA Journal: Model Driven Architecture Straight from the Masters*, Chap. 3, Meghan-Kiffer, Dec. 2004.
3. S. Kelly and J. Tolvanen, "Visual Domain-specific Modeling: Benefits and Experiences of using metaCASE Tools," In *Proc. of Int'l workshop on Model Engineering, ECOOP*, 2000.
4. H. Wada, J. Suzuki, S. Takada and N. Doi, "Leveraging Metamodeling and Attribute-Oriented Programming to Build a Model-driven Framework for Domain Specific Languages," In Proc. of the 8th JSSST Conference on Systems Programming and its Applications, March 2005.
5. H. Wada and J. Suzuki, "Modeling Turnpike Frontend System: a Model-Driven Development Framework Leveraging UML Metamodeling and Attribute-Oriented Programming," In *Proc. of the 8th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, October 2005. to appear.
6. D. Schwarz, "Peeking Inside the Box: Attribute-Oriented Programming with Java 1.5," In *ON Java.com*, O'Reilly Media, Inc., June 2004.
7. Willink, "UMLX: A Graphical Transformation Language for MDA," In *Proc. of OOPSLA*, 2002.
8. Patrascoiu, "Mapping EDOC to Web Services using YATL," In *Proc. of the 8th IEEE International Enterprise Distributed Object Computing Conference*, September 2004.
9. J. Greenfield, K. Short, S. Cook, S. Kent and J. Crupi, "Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools", Wiley, August 2004.
10. S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," In *IEEE Software*, vol. 20, no. 5, Sept./Oct. 2003.
11. Object Management Group, *UML 2.0 Infrastructure Specification*, September, 2003.
12. Business Process Modeling Initiative, *Business Process Modeling Notation (BPMN) 1.0*, May, 2004.
13. ISO/IEC, *Extended Backus-Naur Form*, ISO/IEC 14977, 1996.